Hw3 report

姓名：傅敬倫　　系級：電機四　　學號：b06505011

Problem1. (30%) no collaboration

1.

```
VAE(
  (encoder): Sequential(
    (0): Sequential(
      (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
      (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
    (1): Sequential(
      (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
    (2): Sequential(
      (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
    (3): Sequential(
      (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
    (4): Sequential(
      (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
  )
  (fc_mu): Linear(in_features=2048, out_features=1024, bias=True)
  (fc_var): Linear(in_features=2048, out_features=1024, bias=True)
  (decoder_input): Linear(in_features=1024, out_features=2048, bias=True)
 (decoder): Sequential(
   (0): Sequential(
     (0): ConvTranspose2d(512, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
     (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
     (2): LeakyReLU(negative_slope=0.01)
   )
   (1): Sequential(
     (0): ConvTranspose2d(256, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
     (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
     (2): LeakyReLU(negative_slope=0.01)
   )
   (2): Sequential(
     (0): ConvTranspose2d(128, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
     (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
     (2): LeakyReLU(negative_slope=0.01)
   )
   (3): Sequential(
     (0): ConvTranspose2d(64, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
     (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
     (2): LeakyReLU(negative_slope=0.01)
   )
 )
 (final_layer): Sequential(
   (0): ConvTranspose2d(32, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
   (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
   (2): LeakyReLU(negative_slope=0.01)
   (3): Conv2d(32, 3, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
   (4): Tanh()
 )
)
```

我的 VAE 中分了 encoder 和 decoder。Encoder 中有五層 conv，在 latent space 中的 hidden size = 512，而 decoder 中有五層 convtranspose。

Train：

Optimizer : Adam

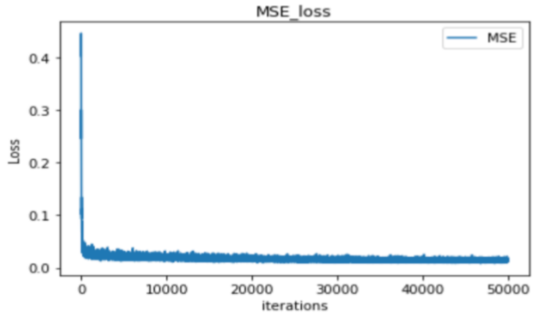
Learning rate : 0.0001, 每 10 個 epoch 下降一半

Loss : MSE and KL with weight 0.0001

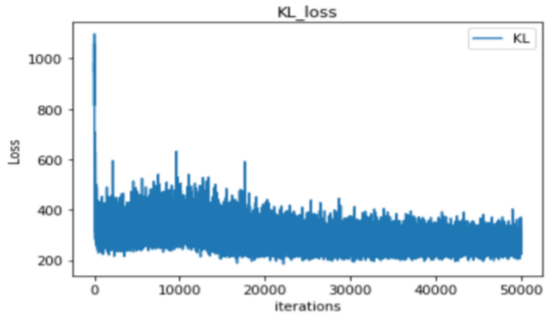
2.

前 50k steps 的 MSE_loss 和 KL_loss

Start to plot!!



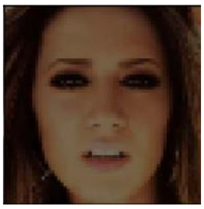
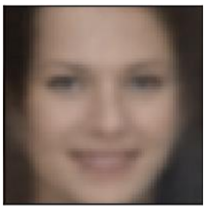
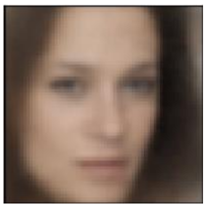
Start to plot!!



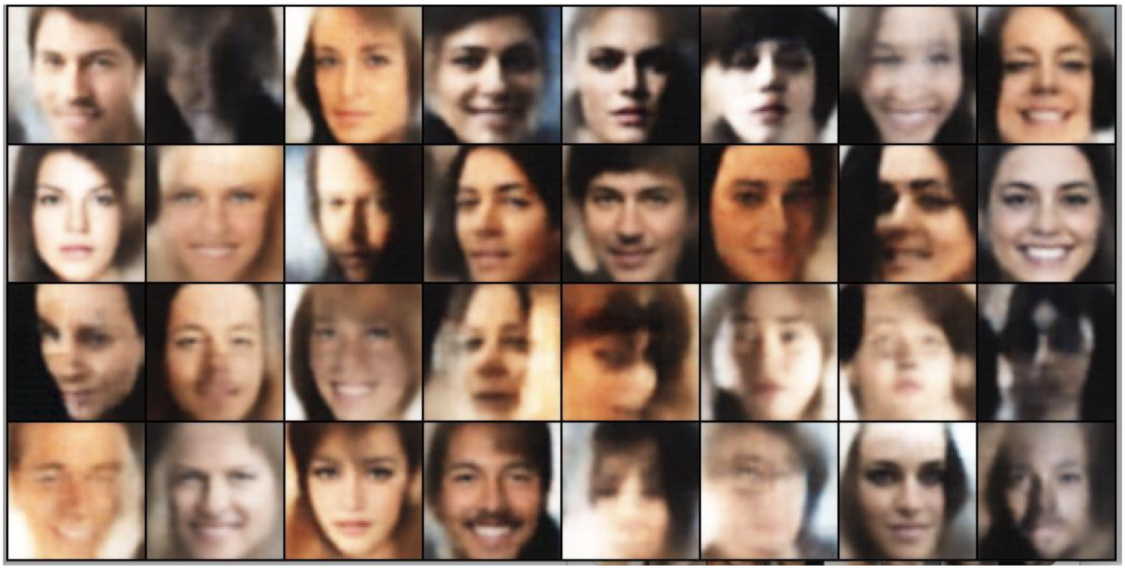
3.

Test Image					
Recon. Image					
MSE	0.0037	0.0037	0.0038	0.0033	0.0037

3.

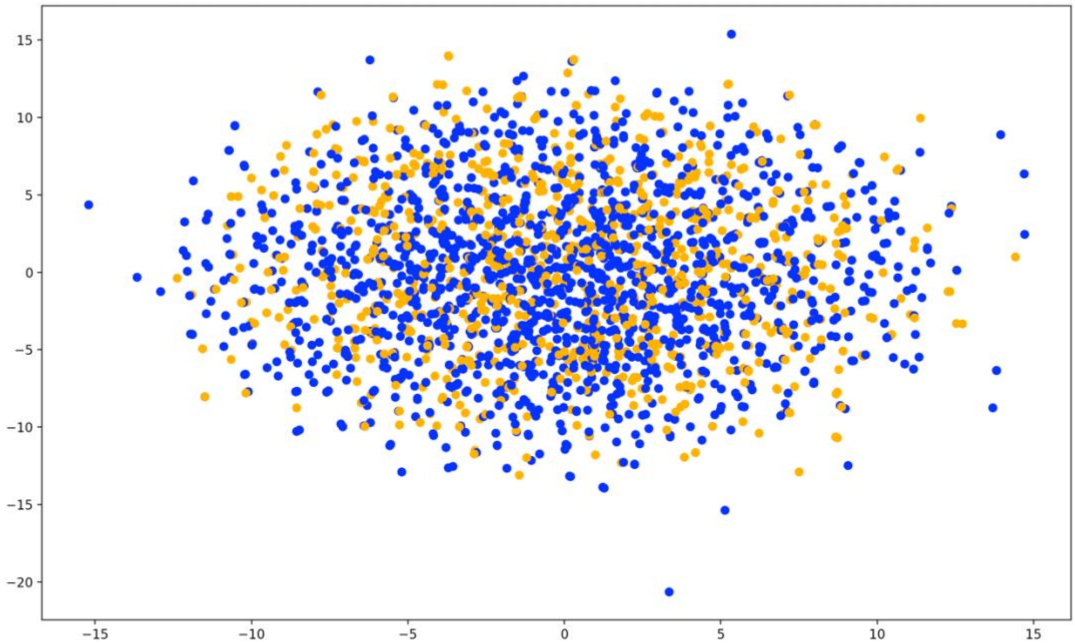
Test Image					
Recon. Image					
MSE	0.0036	0.0027	0.0033	0.0033	0.0034

4.

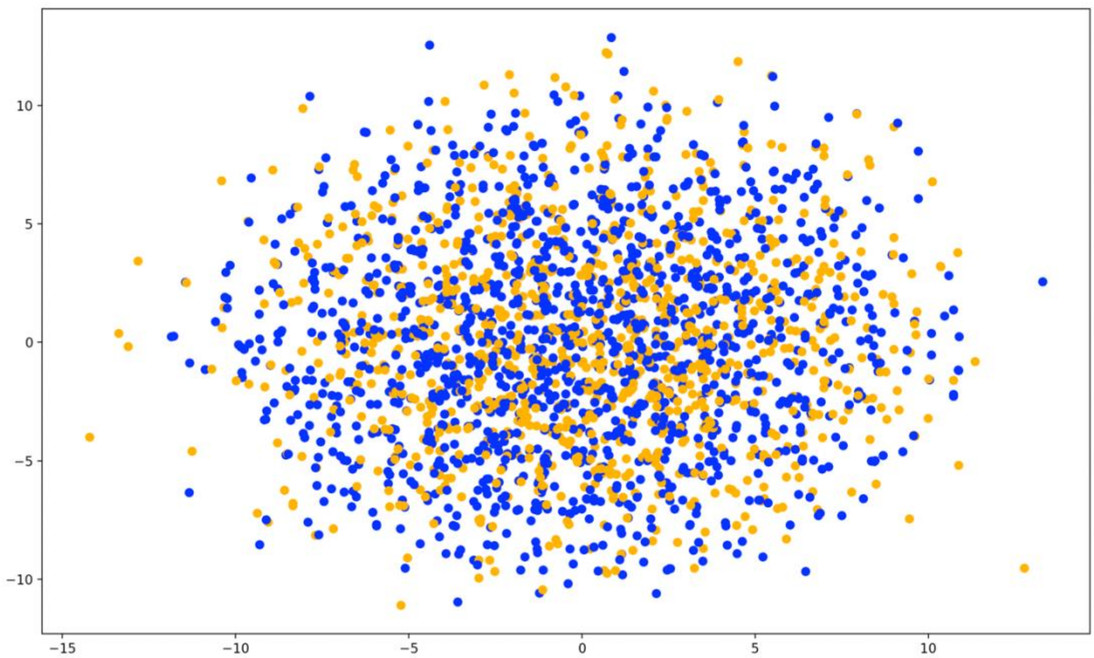


5.

藍色：女生　黃色：男生



藍色：not smiling　　黃色：smiling



6.

在 train VAE 時我有在每一個 epoch 把 Sample 出的圖片印出來（如下圖）：

這是前五個 epoch 我將目前 model 的 sample 圖片印出來，可以發現確實隨著 reconstruct image 的 MSE 越小，Sample 的葡片也越好。但是在後面的 epoch，我發現 reconstruct image 的 MSE 越小，產生出的 Sample 不一定越好，甚至到後面還會越來越扭曲與模糊，因此我在 train VAE 時是每 5 個 epoch 會先看當前產生出的 reconstruct image 和 sample。

Problem2. (20%) no collaboration

1.

```
Generator(
  (main): Sequential(
    (0): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace=True)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
)
Discriminator(
  (main): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (12): Sigmoid()
  )
)
```

Train :

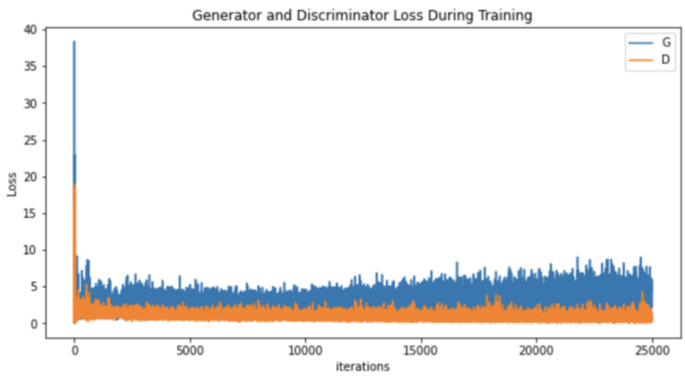
Optimizer : both Adam (G and D)

Learning rate : 0.001

Loss : BCE

Epoch : 10

這是在 train 時，G 和 D 的 loss



2.

3.

在訓練 GAN 的時候，Discriminator 的初始值非常重要，很常如果都猜對的話就會完全 train 不起來，learning rate 也不能調太大很容易就壞掉了。Train GAN 不同於之前的 model 可能 epoch 不能太大，很容易在兩個 model 互相抗衡中爆掉，因此我 GAN 中的 epoch 只用了 5，我也有試過 epoch=10，其實兩者產生出的圖片蠻相似的，沒有特別好。

4.

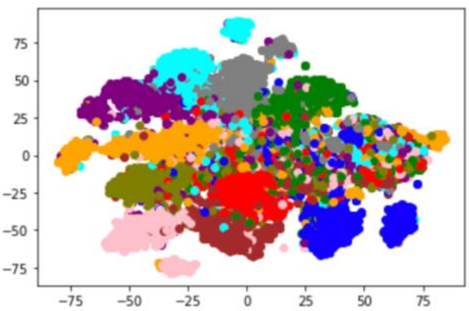
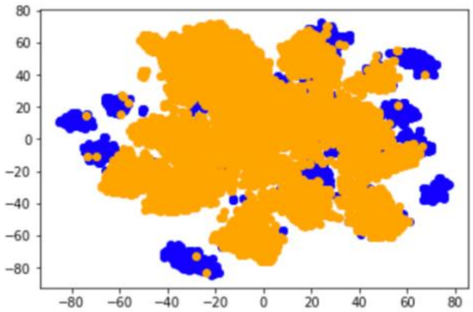
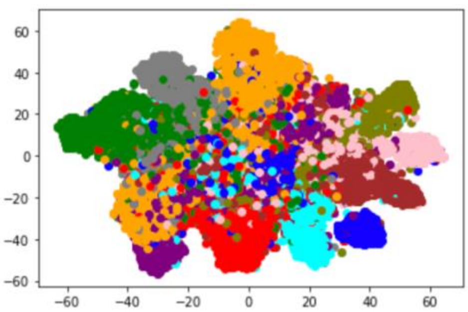
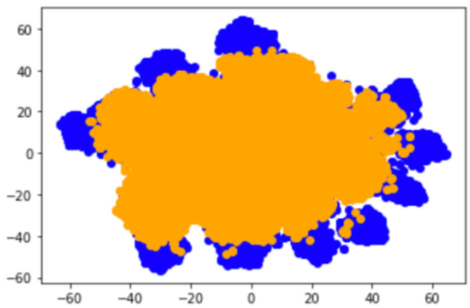
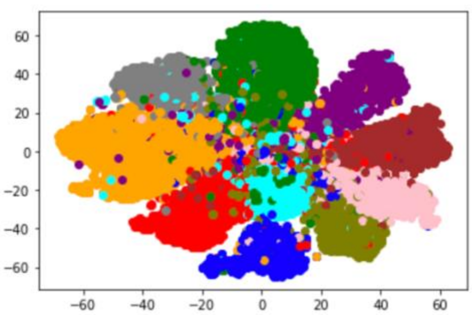
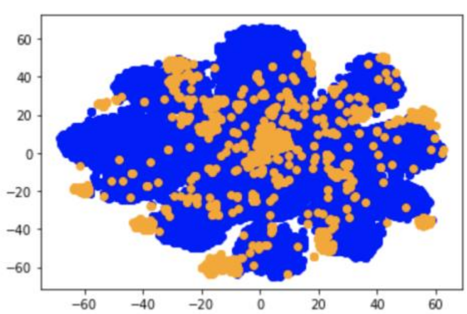
在 VAE 產生出的圖片，可看到比較模糊，但還是看得出臉部的特徵，顏色也比較淡一點。在 GAN 產生的圖片中，整體的顏色更接近實際的膚色，更加亮麗，但是在臉部的特徵上大部分都有一些扭曲。

Problem3. (35%)

1.2.3.

	USPS -> MNIST-M	MNIST-M -> SVHN	SVHN -> USPS
Source only(im+label)	20.8% 20 epoch	34.4% 20 epoch	68.9% 20 epoch
Source(im+label) Target(im)	61% 50 epoch	41.9% 50 epoch	72.1% 100 epoch
Target only(im+label)	98.1% 20 epoch	92.1% 20 epoch	97.3% 20 epoch

4.

	label	domain
U → M		
M → S		
S → U		

5.

```
CNNModel(
  (feature): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU()
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (9): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): ReLU()
    (11): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (12): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (13): ReLU()
    (14): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (15): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (16): ReLU()
  )
  (class_classifier): Sequential(
    (0): Linear(in_features=512, out_features=512, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=512, out_features=10, bias=True)
  )
  (domain_classifier): Sequential(
    (0): Linear(in_features=512, out_features=512, bias=True)
    (1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Linear(in_features=512, out_features=1, bias=True)
  )
)
```

我的 encoder 中會有五個 conv 層，最後會將圖片降到 512 維 (feature size)，class_classifier 會將 feature 分到 10 個類別，domain_classifier 會將 feature 分到一個類別，因為我 domain 的 loss 是使用 BCEWithLogitsLoss，在 forward 中我有用到 ReverselayerF（如下圖）：

```python
class ReverseLayerF(Function):

    @staticmethod
    def forward(ctx, x, alpha):
        ctx.alpha = alpha

        return x.view_as(x)

    @staticmethod
    def backward(ctx, grad_output):
        output = grad_output.neg() * ctx.alpha

        return output, None
```

目的是讓 model 在 backward 時的 gradient 是用減的。

Optimizer	Adam
Learning rate	0.001
Class_loss	CrossEntropyLoss
Domain_loss	BCEWithLogitsLoss

6.

在訓練 DANN 時，我的 Domain_loss 的參數不是像 paper 中寫的 alpha 會隨著 epoch 改變，而是都使用 0.01。在三個任務中，第二個 task（MNISTM→SVHN）是最難的，原本我的 DANN 的 feature 是 512，但我發現在第二個中如果將 feature 加大成 512*2*2 正確率會提高非常多。

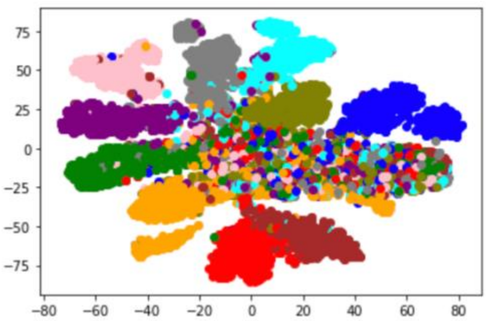
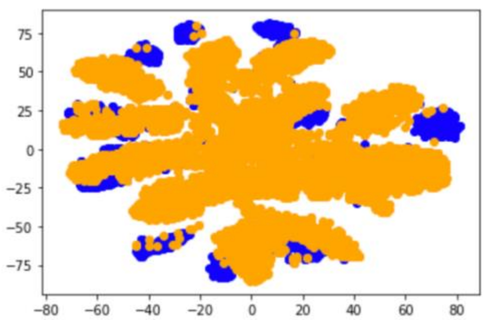
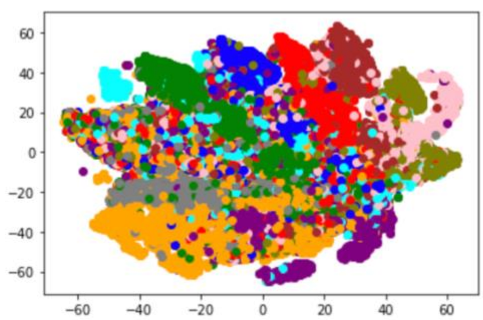
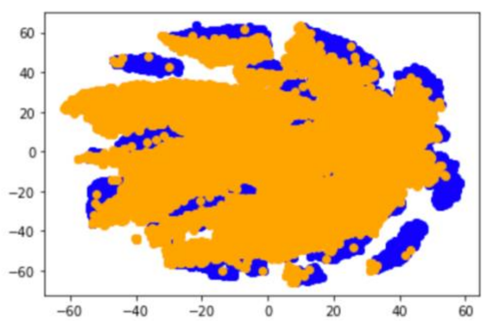
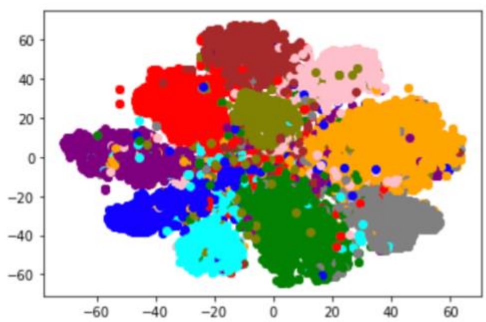
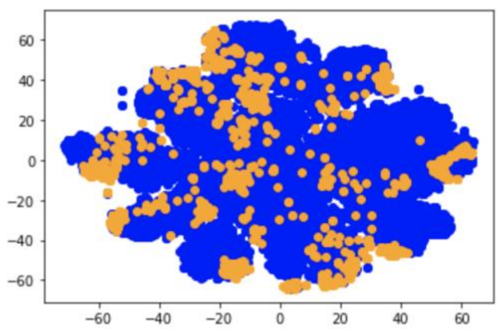
Problem4.

1.

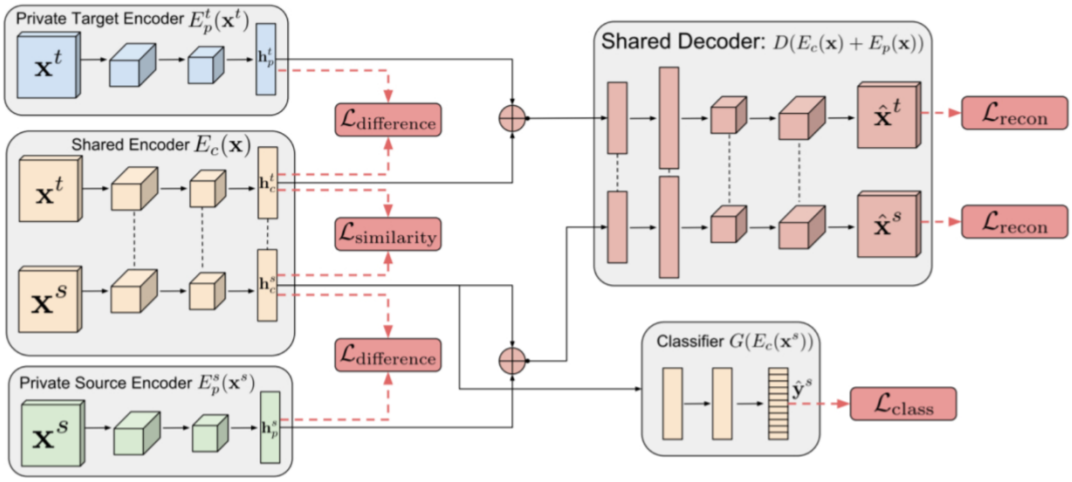
我使用的是 DSN 架構

	USPS -> MNIST-M	MNIST-M -> SVHN	SVHN -> USPS
DSN	68% 100 epoch	52.3% 50 epoch	76.3% 100 epoch
DANN	61% 50 epoch	41.9% 50 epoch	72.1% 100 epoch

2.

	label	domain
U → M		
M → S		
S → U		

3.



我的 improve model 是使用 DSN 架構（如上圖）。我將全部大致上分為六個 model，分別為：Private Target Encoder, Private Source Encoder, Shared Encoder, Shared Decoder, Classifier, Domain Classifier

```
target_encode(
  (target_encoder_conv): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU()
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): ReLU()
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (14): ReLU()
    (15): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (16): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (17): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (18): ReLU()
    (19): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
)

source_encode(
  (source_encoder_conv): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU()
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): ReLU()
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (14): ReLU()
    (15): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (16): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (17): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (18): ReLU()
    (19): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
)
```

```
shared_encode(
  (shared_encoder_conv): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU()
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): ReLU()
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (14): ReLU()
    (15): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (16): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (17): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (18): ReLU()
    (19): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
)
```

```
decoder(
  (layer): Sequential(
    (0): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace=True)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  )
)

classification(
  (shared_encoder_pred_class): Sequential(
    (0): Linear(in_features=512, out_features=512, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=512, out_features=10, bias=True)
  )
)
domainclass(
  (shared_encoder_pred_domain): Sequential(
    (0): Linear(in_features=512, out_features=512, bias=True)
    (1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Linear(in_features=512, out_features=1, bias=True)
  )
)
```

Train:

Optimizer	Adam(6 個 model 都是)
Learning rate	0.0001
Class_loss	CrossEntropyLoss
Domain_loss	BCEWithLogitsLoss
Recon_loss	MSE( ) 自己定義的
Sim_loss	SIMSE( ) 自己定義的
Diff_loss	DiffLoss( ) 自己定義的

```python
class MSE(nn.Module):
    def __init__(self):
        super(MSE, self).__init__()

    def forward(self, pred, real):
        diffs = torch.add(real, -pred)
        n = torch.numel(diffs.data)
        mse = torch.sum(diffs.pow(2)) / n

        return mse


class SIMSE(nn.Module):

    def __init__(self):
        super(SIMSE, self).__init__()

    def forward(self, pred, real):
        diffs = torch.add(real, - pred)
        n = torch.numel(diffs.data)
        simse = torch.sum(diffs).pow(2) / (n ** 2)

        return simse
```

```python
class DiffLoss(nn.Module):

    def __init__(self):
        super(DiffLoss, self).__init__()

    def forward(self, input1, input2):

        batch_size = input1.size(0)
        input1 = input1.view(batch_size, -1)
        input2 = input2.view(batch_size, -1)

        input1_l2_norm = torch.norm(input1, p=2, dim=1, keepdim=True).detach()
        input1_l2 = input1.div(input1_l2_norm.expand_as(input1) + 1e-6)

        input2_l2_norm = torch.norm(input2, p=2, dim=1, keepdim=True).detach()
        input2_l2 = input2.div(input2_l2_norm.expand_as(input2) + 1e-6)

        diff_loss = torch.mean((input1_l2.t().mm(input2_l2)).pow(2))

        return diff_loss
```

在 train 的時候，我是先 train D(Domain Classifier)，讓 D 可以正確的分開 Source 和 Target，接著我會訓練六個 model 就如同 paper 上的架構，這裡的 Domain_loss 我是用減的，讓 Domain 分不清 Source 和 Target，我每一項 Loss 的參數如下圖：

```python
err = err_class + 0.01*err_sim1 + 0.01*err_sim2 + 0.01*err_sim3 + 0.01*err_sim4 + 0.01*err_diff - 0.1 * err_domain
```

4.

在 Train DSN 時，其實比 DANN 來的難非常多，主要是太多 Loss，而每個 Loss 的比例又很難抓，因此一開始 train 的時候，我先將 Loss 只用 Class_loss 和 Domain_loss，Domain_loss 參數跟 DANN 的一模一樣都是 0.1，觀察正確率是否會和 DANN 差不多。接著我再將 Recon,Sim,Diff 加入，去調三個的參數，為了方便三個都用依樣的倍率，最後我是使用 0.01。在訓練時我發現 Diff_loss 幾乎都等於 0，learning rate 在大於 0.01 是完全 train 不起來。在加入額外三個 loss 後，可以發現確實會比 DANN 來得更好一些，Private 和 shared 之間的 loss 確實是有讓 shared encode 完的 feature 保留更多跨 domain 的資訊